19 | 图像分割（上）：详解图像分割原理与图像分割模型

2021-11-24 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远

时长 11:02 大小 10.11M

你好，我是方远。

在前两节课我们完成了有关图像分类的学习与实践。今天，让我们进入到计算机视觉另外一个非常重要的应用场景——图像分割。

你一定用过或听过腾讯会议或者 Zoom 之类的产品吧? 在进行会议的时候，我们可以选择对背景进行替换，如下图所示。

图片来源：https://tech.qq.com/a/20200426/002647.htm

在华为手机中也曾经有过人像留色的功能。

图片来源：https://www.sohu.com/a/294693393_264578

这些应用背后的实现都离不开今天要讲的图像分割。

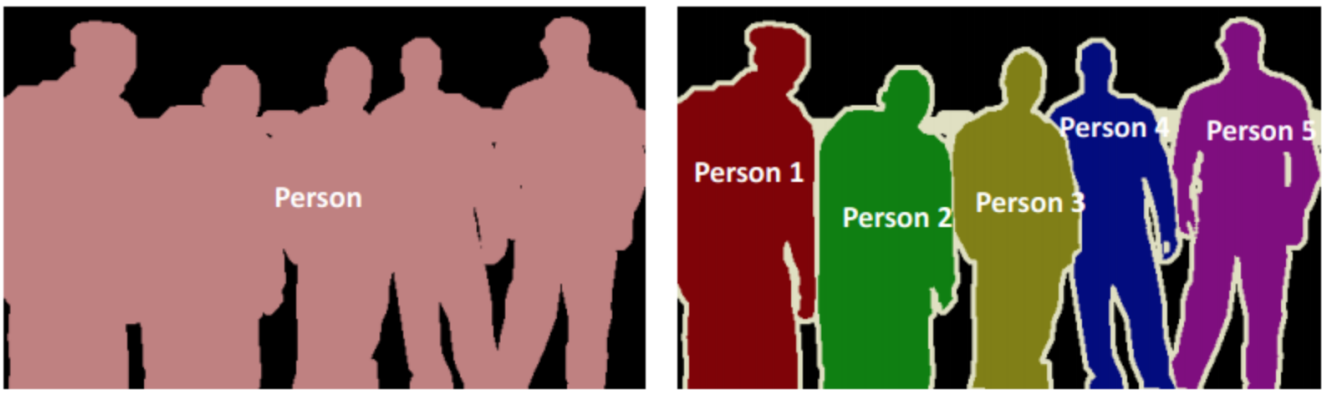
我们同样用两节课的篇幅进行学习，这节课主攻分割原理，下节课再把这些技能点活用到实战上，从头开始搭建一个图像分割模型。

图像分割

我们不妨用对比的视角，先从概念理解一下图像分割是什么。图像分类是将一张图片自动分成某一类别，而图像分割是需要将图片中的每一个像素进行分类。

图像分割可以分为语义分割与实例分割，两者的区别是语义分割中只需要把每个像素点进行分类就可以了，不需要区分是否来自同一个实例，而实例分割不仅仅需要对像素点进行分类，还需要判断来自哪个实例。

如下图所示，左侧为语义分割，右侧为实例分割。我们这两节课都会以语义分割来展开讲解。

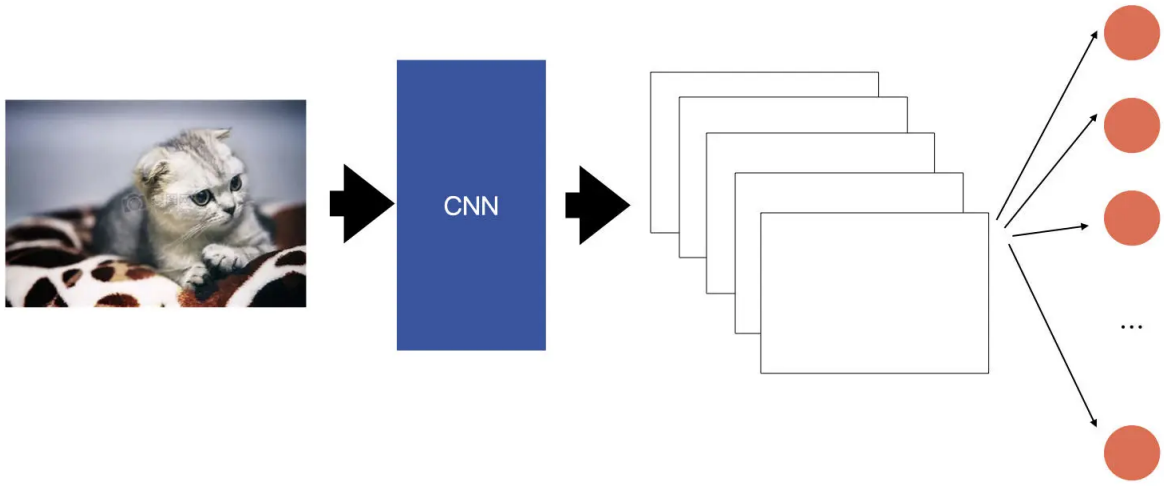


语义分割原理

语义分割原理其实与图像分类大致类似，主要有两点区别。首先是分类端（这是我自己起的名字，就是经过卷积提取特征后分类的那一块）不同，其次是网络结构有所不同。先看第一点，也就是分类端的不同。

分类端

我们先回想一下图像分类的原理。你可以结合下面的示意图做理解。

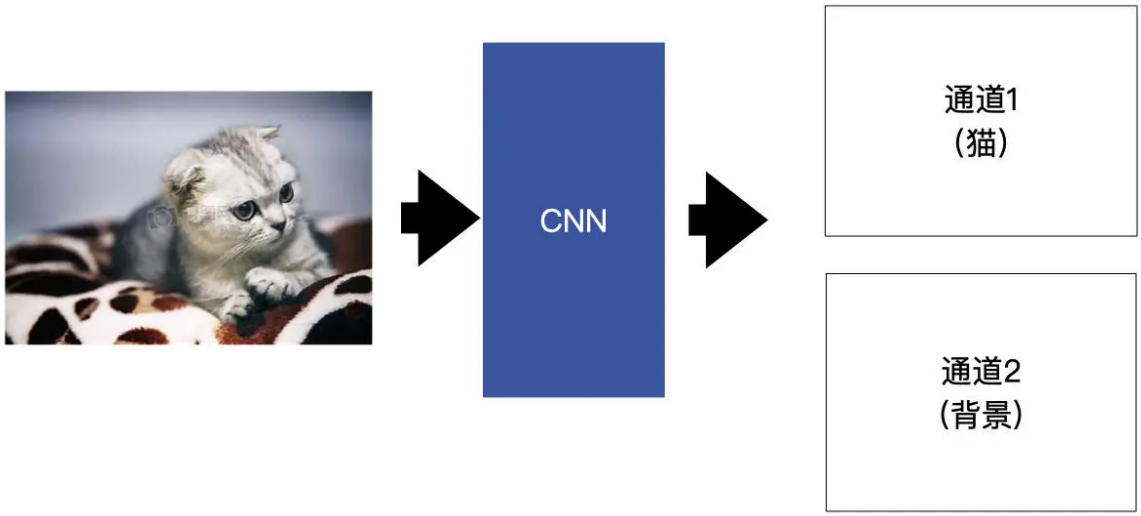


极客时间

输入图片经过卷积层提取特征后，最终会生成若干特征图，然后在这些特征图之后会接一个全连接层（上图中红色的圆圈），全连接层中的节点数就对应着要将图片分为几类。我们将全连接层的输出送入到 softmax 中，就可以获得每个类别的概率，然后通过概率就可以判断输入图片属于哪一个类别了。

在图像分割中，同样是利用卷积层来提取特征，最终生成若干特征图。只不过最后生成的特征图

的数目对应着要分割成的类别数。举一个例子，假设我们想要将输入的小猫分割出来，也就是说，这个图像分割模型有两个类别，分别是小猫与背景，如下图所示。



最终的两个特征图中，通道 1 代表的小猫的信息，通道 2 对应着背景的信息。

这里我给你再举一个例子，来说明一下如何判断每个像素的类别。假设，通道 1 中（0,0）这个位置的输出是 2，通道 2 中（0,0）这个位置的输出是 30。

经过 softmax 转为概率后，通道 1（0, 0）这个位置的概率为 0，而对应通道 2 中 (0,0) 这个位置的概率为 1，我们通过概率可以判断出，在（0，0）这个位置是背景，而不是小猫。

网络结构

在分割网络中最终输出的特征图的大小要么是与输入的原图相同，要么就是接近输入。

这么做的原因是，我们要对原图中的每个像素进行判断。当输出特征图与原图尺寸相同时，可以直接进行分割判断。当输出特征图与原图尺寸不相同时，需要将输出的特征图 resize 到原图大小。

如果是从一个比较小的特征图 resize 到一个比较大的尺寸的时候，必定会丢失掉一部分信息的。所以，输出特征图的大小不能太小。

这也是图像分割网络与图像分类网络的第二个不同点，在图像分类中，经过多层的特征提取，最后生成的特征图都是很小的。而**在图像分割中，最后生成的特征图通常来说是接近原图的**。

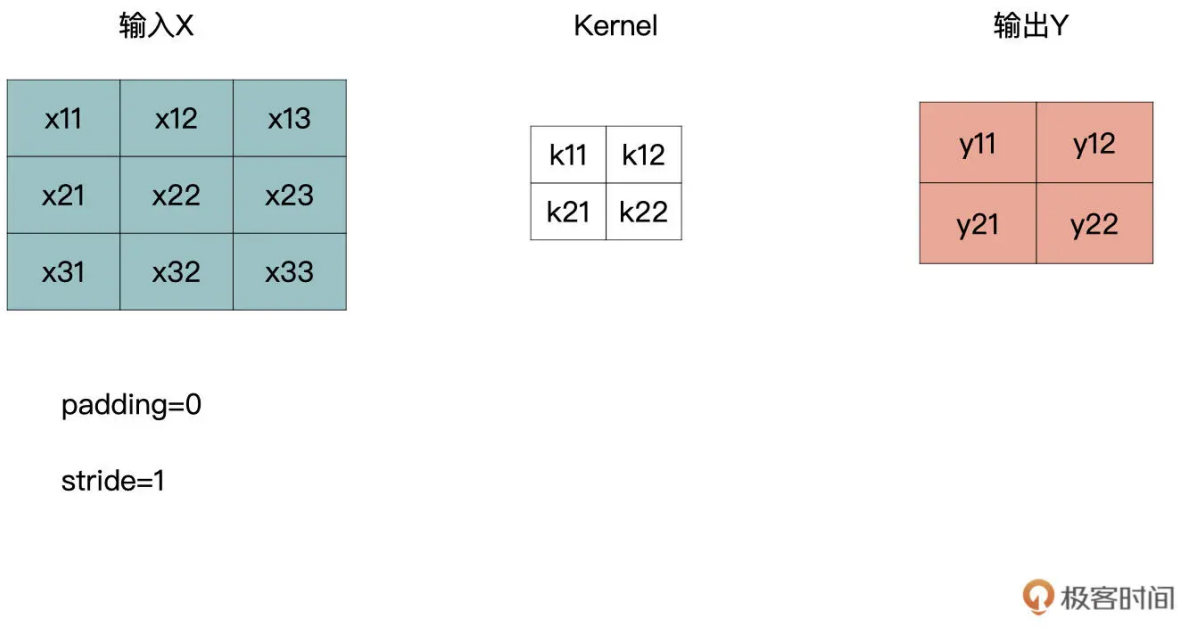
前文也说过，图像分割网络也是通过卷积进行提取特征的，按照之前的理论特征提取后，特征图尺寸是减小的。如果说把特征提取看做 Encoder 的话，那在图像分割中还有一步是 Decoder。

Decoder 的作用就是对特征图尺寸进行还原的过程，将尺寸还原到一个比较大的尺寸。这个还原的操作对应的就是上采样。而在上采样中我们通常使用的是转置卷积。

转置卷积

接下来我就带你研究一下转置卷积的计算原理，这也是这节课的重点内容。

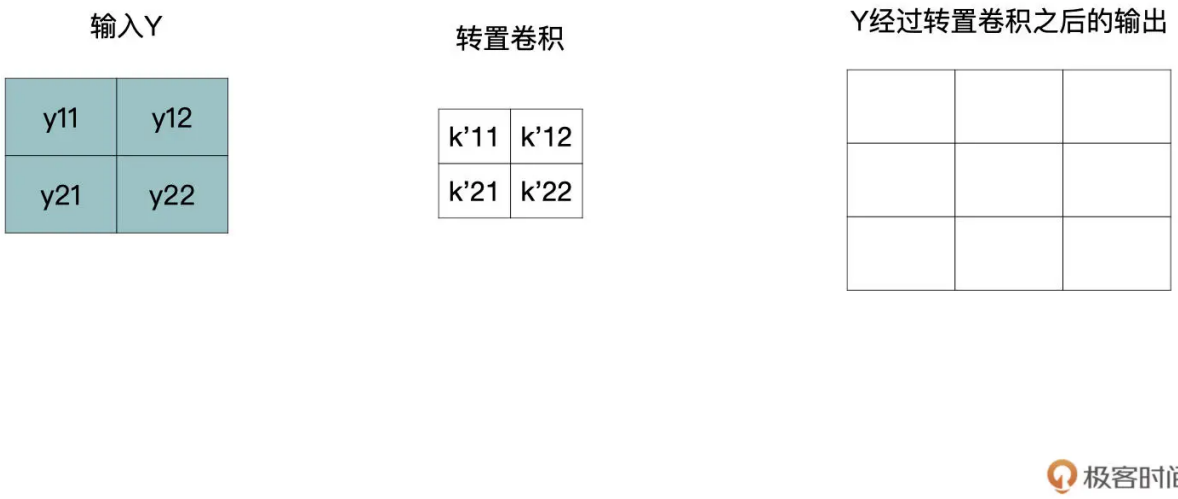
我们看下面图这个卷积计算，padding 为 0，stride 为 1。



从之前的学习我们可以知道，卷积操作是一个多对一的运算，输出中的每一个 y 都与输入中的 4 个 x 有关。其实，转置卷积从逻辑上是卷积的一个逆过程，而**不是卷积的逆运算。**

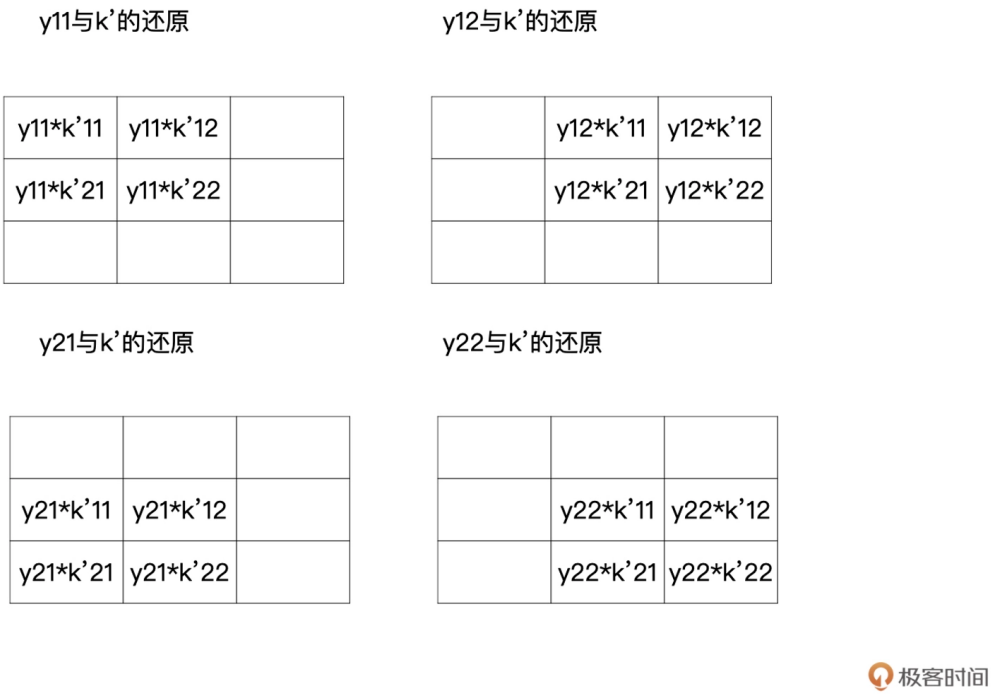
也就是说，转置卷积并不是使用上图中的输出 Y 与卷积核 Kernel 来获得上图中的输入 X，转置卷积只能还原出一个与输入特征图**尺寸**相同的特征图。

我们将转置卷积中的卷积核用 k’ 表示，那么一个 y 会与四个 k’ 进行还原，如下所示：



极客时间

还原尺寸的过程如下所示，下图中每个还原后的结果都对应着原始 3x3 的输入。



极客时间

通过观察你可以发现，有些部分是重合的，对于重合部分把它们加起来就可以了，最终还原后的特征图如下：

y11*k'11	y11*k'12+y12*k'11	y12*k'12
y11*k'21+y21*k'11	y11*k'22+y12*k'21+y21*k'12+y22*k'11	y12*k'22+y22*k'12
y21*k'21	y21*k'22+y22*k'21	y22*k'22

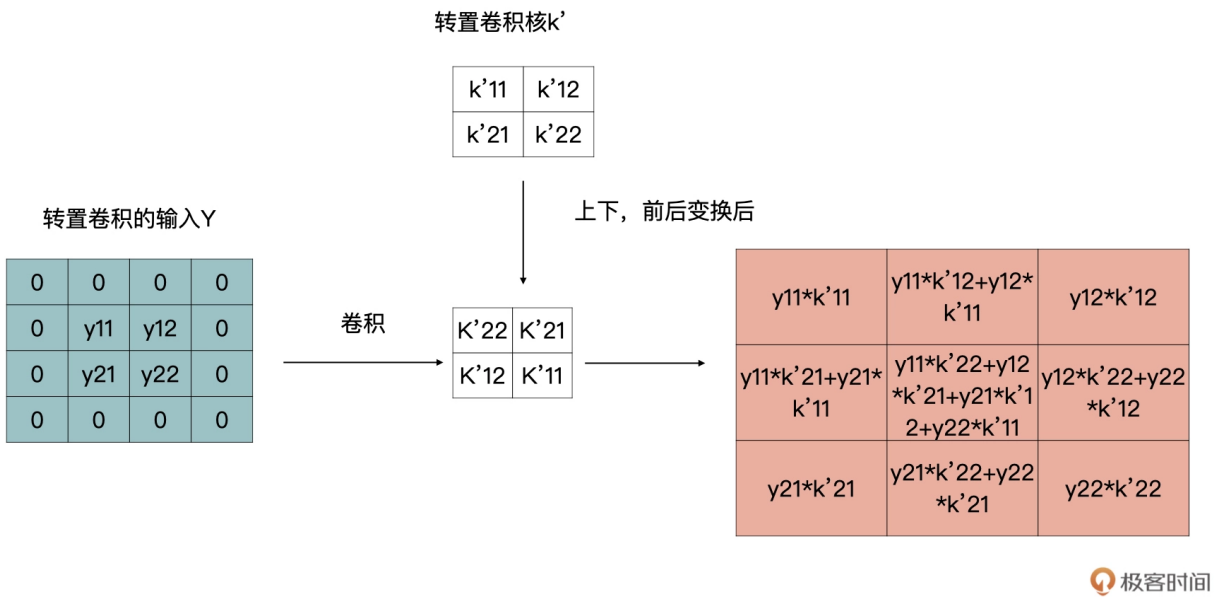
极客时间

将上图的结果稍作整理，整理为下面的结果，也没有做什么特殊处理，只是补了一些零：

0 * k'22 + 0 * k'21 + 0 * k'12 + y11 * k'11	0 * k'22 + 0 * k'21 + y11 * k'12 + y12 * k'11	0 * k'22 + 0 * k'21 + y11 * k'12 + 0 * k'11
0 * k'22 + y11 * k'21 + 0 * k'12 + y21 * k'11	y11 * k'22 + y12 * k'21 + y21 * k'12 + y22 * k'11	y12 * k'22 + 0 * k'21 + y22 * k'12 + 0 * k'11
0 * k'22 + y21 * k'21 + 0 * k'12 + 0 * k'11	y21 * k'22 + y22 * k'21 + 0 * k'12 + 0 * k'11	y22 * k'22 + 0 * k'21 + 0 * k'12 + 0 * k'11

极客时间

上面的结果，我们又可以通过下面的卷积获得：



你有没有发现一件很神奇的事情，转置卷积计算又变回了卷积计算。

所以，我们一起梳理一下，转置卷积的计算过程如下：

1. 对输入特征图进行补零操作。

2. 将转置卷积的卷积核上下、左右变换作为新的卷积核。

3. 利用新的卷积核在 1 的基础上进行**步长为 1**，**padding 为 0** 的卷积操作。

我们先来看一下，PyTorch 中转置卷积以及它的主要参数，再根据参数解释一下第一步 1 是如何补零的。

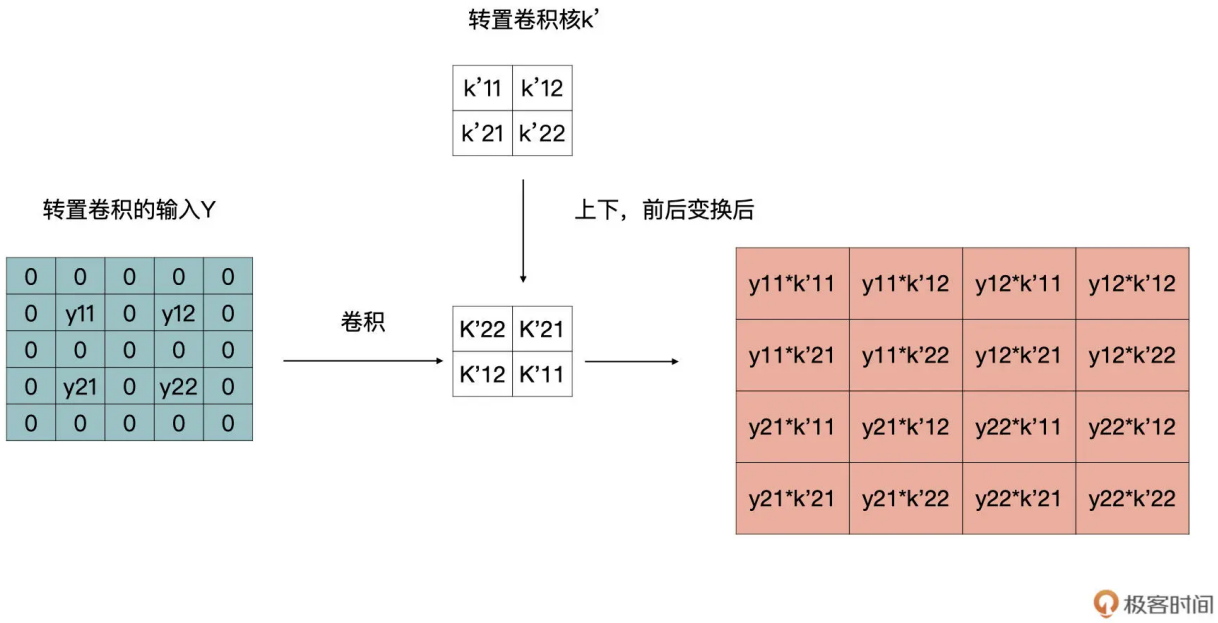
复制代码

```
class torch.nn.ConvTranspose2d(in_channels,
                               out_channels,
                               kernel_size,
                               stride=1,
                               padding=0,
                               groups=1,
                               bias=True,
                               dilation=1)
```

其中，in_channels、out_channels、kernel_size、groups、bias 以及 dilation 与我们之前讲卷积时的参数含义是一样的（你可以回顾卷积的第9、10两节课），这里我们就不赘述了。

首先，我们看一下 stride。因为转置卷积是卷积的一个逆向过程，所以这里的 stride 指的是在原图上的 stride。

在我们刚才的例子里，stride 是等于 1 的，如果等于 2 时，按照同样的套路，可以转换为如下的卷积变换。 同时，我们也可以得到结论，上文中第一步，补零的操作是，在输入的特征图的行与列之间补 stride-1 个行与列的零。



再来看 padding 操作，padding 是指要在输入特征图的周围补 dilation * (kernel_size - 1) - padding 圈零。这里用到了 dliation 参数，但是通常在转置卷积中 dilation、groups 参数使用的比较少。

以上就是转置卷积的补零操作了，图片和文字双管齐下，我相信你一定能够理解它。

通过上述的讲解，我们可以推导出输出特征图尺寸与输入特征图尺寸的关系：

$$h_{out} = (h_{in} - 1) * stride[0] - padding[0] + kernel\_size[0]$$

$$w_{out} = (w_{in} - 1) * stride[1] - padding[1] + kernel\_size[1]$$

下面，我们借助代码来验证一下，我们讲的转置卷积是否是向我们所说的那样计算。

现在有特征图 input_feat:

复制代码

```
import torch
import torch.nn as nn
import numpy as np
input_feat = torch.tensor([[[[1, 2], [3, 4]]]], dtype=torch.float32)
input_feat
输出：
tensor([[[[1., 2.],
          [3., 4.]]]])
```

卷积核 k：

复制代码

```
kernels = torch.tensor([[[[1, 0], [1, 1]]]], dtype=torch.float32)
kernels
输出：
tensor([[[[1., 0.],
          [1., 1.]]]])

```

stride 为 1，padding 为 0 的转置卷积：

复制代码

```
convTrans = nn.ConvTranspose2d(1, 1, kernel_size=2, stride=1, padding=0, bias
convTrans.weight=nn.Parameter(kernels)
```

按照我们刚才讲的，第一步是补零操作，输入的特征图补零后为：

$$input\_feat = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \end{bmatrix}$$

然后再与变换后的卷积核：

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

做卷积运算后，获得输出：

$$output = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 7 & 2 \\ 3 & 7 & 4 \end{bmatrix}$$

我们再看看代码的输出，如下所示：

复制代码

```
convTrans(input_feat)
输出：
tensor([[[[1., 2., 0.],
          [4., 7., 2.],
          [3., 7., 4.]]]], grad_fn=<SlowConvTranspose2DBackward>)
```

你看看是不是一样呢？

损失函数

说完网络结构，我们再开启图像分割里的另一个话题：损失函数。

在图像分割中依然可以使用在图像分类中经常使用的交叉熵损失。在图像分类中，一张图片有一个预测结果，预测结果与真实值就可以计算出一个 Loss。而在图像分割中，真实的标签是一张二维特征图，这张特征图记录着每个像素的真实分类结果。在分割中，含有像素类别的特征图，我们一般称为 Mask。

我们结合一张小猫图片的例子解释一下。对于下图中的小猫进行标记，标记后会生成它的 GT，这个 GT 就是一个 Mask。

GT 是 Ground Truth 的缩写，在图像分割中我们经常使用这个词。在图像分类中与之对应的就是数据的真实标签，在图像分割中则 GT 是每个像素的真实分类，如下面的例子所示。

GT 如下所示：

那在我们模型预测的 Mask 中，每个位置都会有一个预测结果，这个预测结果与 GT 中的 Mask 做比较，然后会生成一个 Loss。

当然，在图像分割中不光有交叉熵损失可以用，还可以用更加有针对性的 Dice Loss，下节课我再继续展开。

公开数据集

刚才我们也看到了，图像分割的数据标注还是比较耗时的，具体如何标注一张语义分割所需要的图片，下节课我们再一起通过实践探索。

除此之外业界也有很多比较有权威性且质量很高的公开数据集。最著名的就是 COCO 了，链接如下：https://cocodataset.org/#detection-2016。一共有 80 个类别，超过 2 万张图片。感兴趣的话，课后你可以尝试着使用它训练来看看。

小结

恭喜你完成了今天的学习。

今天我们首先明确了语义分割要解决的问题是什么，它可以对图像中的每个像素都进行分类。

然后我们对比图像分类原理，说明了语义分割的原理。它与图像分类主要有两个不同点：

1. 在分类端有所不同，在图像分类中，经过卷积的特征提取后，最后会以若干个神经元的形式作为输出，每个神经元代表着对一个类别的判断情况。而语义分割，则是会输出若干的特征图，每个特征图代表着对应类别判断。

2. 在图像分类的网络中，特征图是不断减小的。但是在语义分割的网络中，特征图还会有 decoder 这一步，它是将特征图进行放大的过程。实现 decoder 的方式称为上采样，在上采样中我们最常使用的就是转置卷积。

对于转置卷积，我们除了要知道它是怎么计算的之外，最重要的是要记住**它不是卷积的逆运算，只是能将特征图的大小进行放大的一种卷积运算**。

每课一练

对于本文的小猫分割问题，最终只输出 1 个特征图是否可以？

欢迎你在留言区跟我交流互动，也推荐你把这节课分享给更多同事、朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | 图像分割（下）：如何构建一个图像分割模型？

下一篇 20 | 图像分割（下）：如何构建一个图像分割模型？

更多学习推荐

精选留言 (3)

写留言

王琮
2021-11-24

只输出一个特征图应该是可以的，输出主体-猫，剩下的部分即为背景，或者输出背景，剩下的部分即是猫。

展开

作者回复: hi，王琮。是可以的，
因为小猫分割是一个二分类问题，可以将输出的特征图使用sigmoid函数将输出的数值转换为一个概率，从而进行判断。

官
2021-11-24

可以的，只要把通过概率判断是小猫的像素输出即可。

作者回复: ^^ 👍👍

文森特没有眼泪

2021-11-24

我认为可以，使用 sigmoid 处理后可以把大于 0.5 的像素认为是小猫，小于 0.5 的像素认为是背景。

作者回复: 👍👍^^，是的，可以的。